

Relazione II Progetto

Laboratorio di Algoritmi e Strutture Dati

Gabriele Savio, Michele Di Cosmo

Premessa:

I dati utili al fine dell'esperimento sono stati raccolti su una macchina con processore Pentium M 1.73 Ghz 2Gb di RAM utilizzando Il Sistema Operativo Windows XP SP2. Al fine dell'esperimento si è deciso di utilizzare un numero di grafi pari a 50.

Per quanto riguarda l'analisi statistica si è fatto riferimento alla parte studiata durante la prima parte del corso: per il numero di ripetizioni di ogni esperimento si è deciso di utilizzare un errore di misurazione del 5% e un coefficiente di confidenza pari al 5%.

Obiettivo

Il nostro obiettivo era quello di misurare e mettere a confronto i tempi medi di esecuzione degli algoritmi di Prim e di Kruskal per la ricerca della foresta di costo minimo per un grafo.

Gli Algoritmi

L'algoritmo di Kruskal è stato implementato con il codice visto a lezione. L'algoritmo prima di tutto crea $|V|$ alberi (utilizzando le classi DisjointTreeHeuristic, DisjointTreeNode, Link), quindi ordina tutti gli archi del grafo in senso decrescente rispetto al peso (utilizzando Edge e HeapSortEdge), quindi per ogni arco partendo da quello col peso minore se i 2 rappresentanti dei nodi di quest'arco sono diversi allora li unisce.

L'algoritmo complessivamente opera in un tempo $O(E \lg V)$ (DisjointTreeHeuristic opera con UnionByRank e PathCompression).

L'algoritmo di Prim, anch'esso implementato con il codice del corso, inizia da un arbitrario vertice (nel nostro caso 0) e crea una MinHeap (con la classe AdjHeap).

Ad ogni passo estrae il minimo dalla heap, aggiunge il nodo all'albero, cambia se necessario i valori dei figli del nodo estratto e ripristina la heap.

L'algoritmo esegue asintoticamente a Kruskal in un tempo $O(E \lg V)$.

Analisi degli Esperimenti

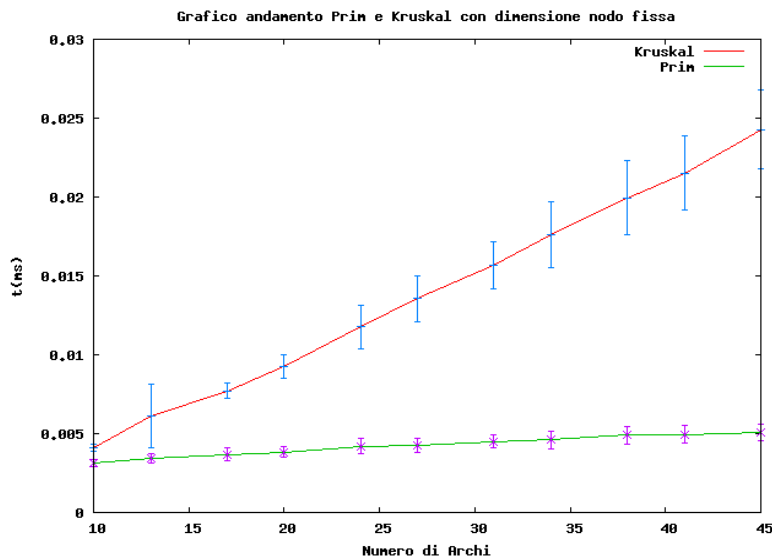


Figura 1 - Grafico con 10 nodi

L' algoritmo di Prim risulta quindi più efficiente.

Come si può notare dai risultati ottenuti l' algoritmo di Kruskal risulta decisamente più lento rispetto all' algoritmo di Prim.

Inoltre l' algoritmo di Kruskal presenta tempi molto vari rispetto a Prim. Quindi la nostra implementazione di Kruskal risente molto più di Prim della casualità dei grafi.

Nel caso di un numero esiguo di archi, i due algoritmi eseguono quasi nello stesso tempo, mentre all' aumentare degli archi divergono in modo marcato.

Possiamo scoprire il motivo di questa differenza analizzando meglio la complessità dell' algoritmo di Kruskal: considerando il variare della dimensione degli archi con $|V| \leq |E| \leq \frac{(|V|(|V|-1))}{2}$ possiamo trovare un riscontro nella teoria alle nostre analisi.

Nell' algoritmo di Kruskal abbiamo innanzitutto il costo $O(E \lg E)$ per l' ordinamento degli archi, da noi realizzato tramite HeapSort. Notiamo che il tempo delle operazione con gli insiemi disgiunti è $O(E \alpha(V))$ che è approssimabile a $O(E \lg E)$ il che spiega il motivo per cui l' algoritmo di Kruskal è molto più sensibile alla variazione della

dimensione degli archi: il termine E (che indica la dimensione degli

archi) è decisamente più incisivo della dimensione del nodo nella complessità dell' algoritmo.

Al contrario, nella complessità dell' algoritmo di Prim il numero di archi si trova solo nell' approssimazione del numero di passi $O(E)$ che l' algoritmo esegue.

Di conseguenza possiamo concludere che anche teoricamente l' incidenza del numero di archi nell' algoritmo di Kruskal è decisamente più marcata che nell' algoritmo di Prim.

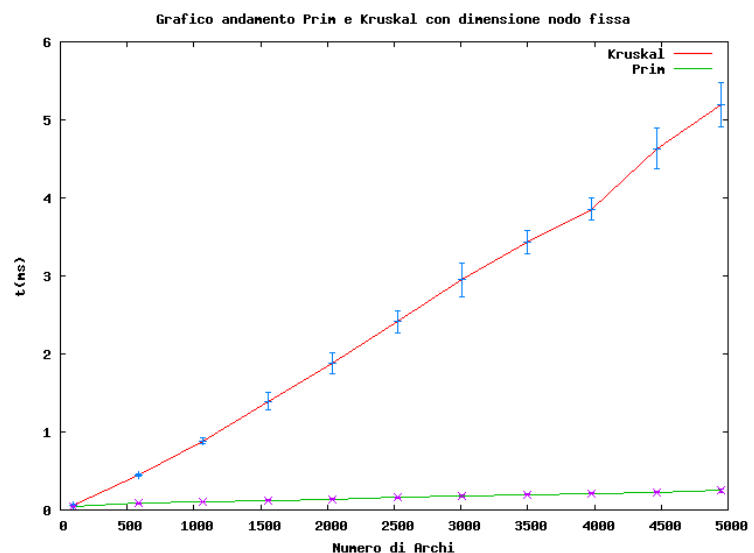


Figura 2 - Grafico con 100 nodi

In conclusione i dati raccolti mostrano l'andamento prevedibile dei due algoritmi e mostrano la sostanziale ininfluenza del numero di nodi come fattore nel determinare il tempo di esecuzione dell'algoritmo.

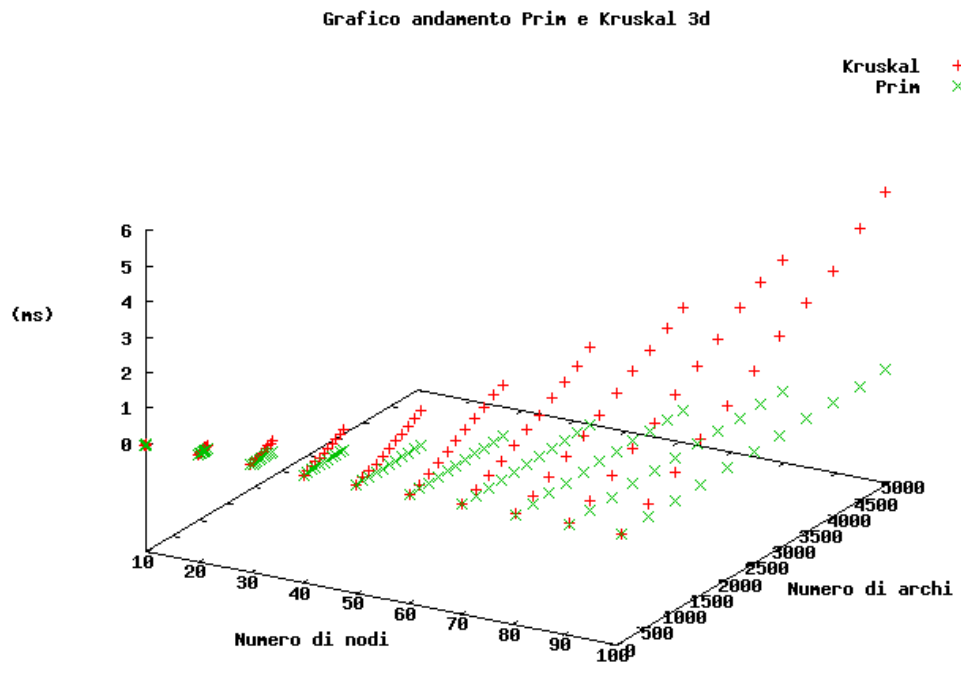


Figura 3 – Visuale 3D